Realtime Collision Detection for Virtual Reality Applications *

Ji-Hoon Youn and K. Wohn

Center for Artificial Intelligence Research
Computer Science Department
Korea Advanced Institute of Science & Technology
Taejon, Korea

Abstract

Narrow bandwidth between computers and humans limits the usage of computers up to their potential. The virtual reality technology aims at the expansion of the communication bandwidth by providing users with three dimensional immersive environments. For the true direct manipulation of the environments, fast collision detection must be provided to increase the sense of reality. In this paper we propose a collision detection scheme for virtual reality applications. The method exploits a hierarchical object representation to facilitate the detection of colliding segments.

1 Introduction

VR(Virtual Reality) technology allows the computer users to enter the computer-generated virtual world, and interact with graphical objects and virtual agents with the sense of reality. The physical interactions such as touching, hitting, throwing, etc, for the most part, are triggered by collision. Therefore, in order to increase the sense of reality for a VR system, it is necessary to detect collision between the graphical objects in realtime.

There has been much research on the collision detection problem in the related fields such as computer graphics and robotics [1, 2, 3, 4, 5, 6, 8, 9]. But the existing methods are not suitable to the realtime application as is ours. Our approach to achieve the realtime performance is to localize the possible collision regions by utilizing a hierarchical representation scheme for 3D objects. In this way we can detect the collision fast because there may be simpler geometrical objects in the localized region. Thus, we concentrate more on the localization method than on the collision detection problem itself.

In this paper, we present the collision localization algorithm between two hierarchical objects. As the data structure that supports the algorithm, the C-tree is introduced. Its usage facilitates localizing the collision region effectively. We have implemented this idea so that we manipulate 3D objects directly in realtime with the 3D glove-cursor which is controlled by the hand motion.

*This work is partially supported by the grants from CAIR, KOSEF and KIST.

0-7803-1363-1/93 $3.00 © 1993 IEEE

2 Collision Localization Algorithm for Hierarchical Objects

2.1 Hierarchical Objects

In our domain of virtual world, the graphical objects are assumed to be represented as the hierarchical structure. A *Hierarchical Object*(HO) is composed of multiple segments. Each segment is related to others such that the object as a whole is represented as the tree structure. In our implementation, sibling relations are added for performing iterative operations efficiently and for avoiding multiple links from one parent to its children. Moreover, the movement of each segment may affect the positions and orientations of its entire descendants in the world coordinate system. Figure 1(a) shows a simple human HO. Figure 1(b) shows the relations between segments, which indicates that the root is the hip h. The value of pointers without a link is null.

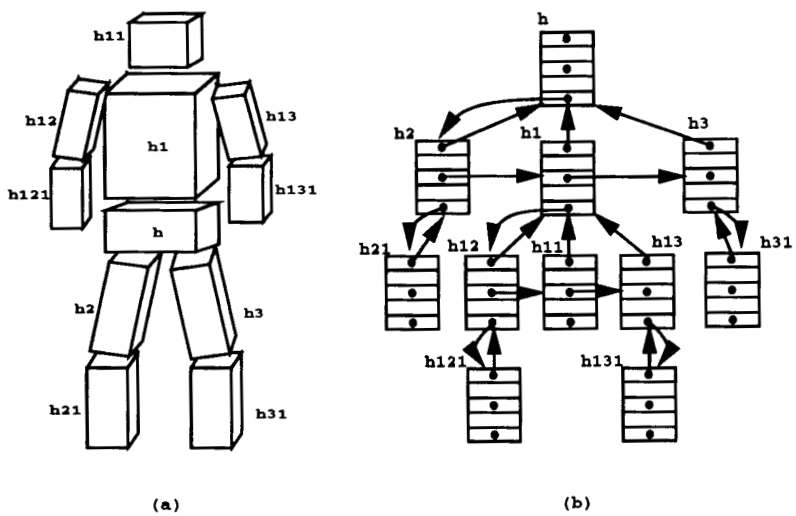

Figure 1: Hierarchical object.

2.2 The C-Tree

We devise the *C-Tree* to detect collisions between hierarchical objects efficiently. The association between C-tree and HO is made by the pointers from C-tree's node to HO's segment.

The C-Tree is formally denoted by a tree (V,E), where V is a set of nodes and E is a relation on V, such that for any node n in V,

$$\bigcup\{a \mid a \text{ is the child of } n\} \subseteq n \tag{1}$$

must hold in time. The basic idea of C-tree is that each node of C-tree has the shape enclosing all of the child nodes such that non-leaf nodes take the role of localizing possible collision regions, and the leaf nodes have actual shapes to be used for the collision detection.

The C-tree has the following structure: the nodes with a pointer to HO's segment have the shape corresponding to the pointed segment. When a node has a null-valued pointer, its shape is determined by the predetermined method. Note that all leaf nodes of C-trees should have the non-null pointer since these leaf nodes have the actual shapes for collision detection. When some segments of a HO have moved in the world coordinate system, all ancestors of its C-tree's nodes relevant to these segments should change their shape. The shapes of the nodes with null-valued pointer are automatically calculated whenever their shapes should change so that they maintain the bounding spheres enclosing the children's shapes. The shape for any node of C-trees is confined to sphere or convex polyhedron. This limitation is not too restrictive because concave segments can be represented by the finite union of convex segments. Moreover the collision models of very complex visual segments may be approximated by simple invisible segments.

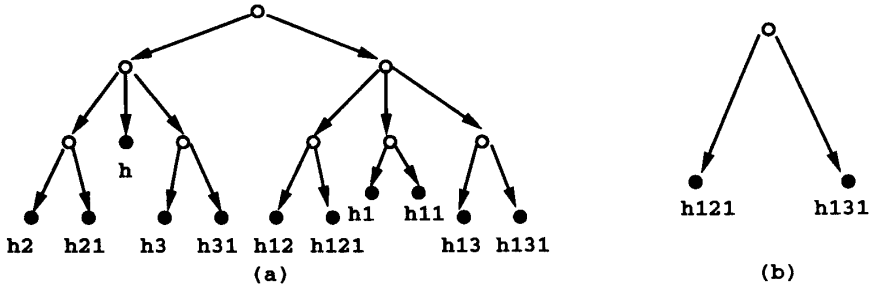

Figure 2: Possible C-trees of the human hierarchical object.

Figure 2 depicts two of possible configurations of C-trees for the human object of figure 1(a). The label of the filled nodes indicates values of the pointer to the segment of the human hierarchical object. The nodes represented as an empty circle denote that their pointer values are null. Figure 2(a) shows one of the most detailed C-tree configurations of the human HO. The configuration in figure 2(b) is appropriate for the applications interested in the hands of the human model collision detection. For example, the human object interacting with other objects only by using the hands.

In summary, the collision properties of HO can be represented in terms of its C-tree. The main role of C-tree is to localize the collision regions between hierarchical objects as quickly as possible. Although the efficiency of localization of collision region depends on the proper configuration of C-tree and this configuration requires some additional work, the C-tree leads us to adopt a natural collision detection algorithm and to get the real time performance from its fast localization. Since the collision property of HO can be represented by its C-tree, the problem of collision detection between hierarchical objects becomes the problem of collision detection between the C-trees. In Section 2.3, the algorithm for collision detection between the C-trees is presented.

2.3 Collision Localization Algorithm between C-Trees

The collision detection algorithm between two C-trees has two phases: the first is to create the list of highly collidable pairs among the leaf nodes. The second phase determines whether two shapes of each pair in the list collide or not. The C-like pseudo code for the first phase algorithm is given as follows:

```
localize(Ctree a, Ctree b)
{
   if (both a and b are leaf nodes)
   {
      insert(a,b);
      return;
   }

   if (!collide(a,b)) return;

   if (a is a leaf node)
   {
      for (all child b_i of b) localize(a,b_i);
   }
   else if (b is a leaf node)
   {
      for (all child a_i of a) localize(a_i,b);
   }
   else
   {
      for (all child a_i of a)
      {
         if (!collide(a_i,b)) continue;
         for (all child b_i of b) localize(a_i,b_i);
      }
   }
}
```

Localize() recursively creates the list of all collidable pairs from C-trees' leaf nodes. *Insert*() handles this list. After *collide*() solves the collision detection problem according to the shapes of given parameters it returns true or false. When the first phase has been completed, the list contains the pairs of leaf nodes which are likely to collide each other. The reason to create the list is that we can post-process it so that we may discard the uninterested pairs quickly. The second phase is trivial. While scanning the list, it reports all pairs which really collide each other. Note that there may be other possible variants of localize() for more speed-up. This algorithm just shows the basic skeleton.

Our algorithm does not detect the collision among the segments of single HO h. However, this is trivial to solve because, when needed, localize(h,h) will detect the collision completely. Note that the pairs consisting of the same segment in the second phase will be discarded from the list to avoid a meaningless collision detection operation.

Until now, we have considered the collision detection problem between two hierarchical objects. We, however, must detect all pairs of HO's when there are n HO's in the virtual world. When we consider all pairs, it needs $O(n^2)$ pairwise collision detection. To avoid this complexity, it is also needed to localize the collision regions so that we may test only the pairs which are highly probable to collide each other in the localized regions. Although not implemented yet, this problem can be tackled by utilizing the C-tree as well. Main

idea is to construct a universal C-tree u such that its leaf nodes are composed of the root nodes of all C-trees. Then, localize(u,u) will detect all collidable pairs and we can conduct the collision detection on these highly collidable pairs only.

The efficiency of the collision localization algorithm depends highly on the C-tree's configuration. For a fast localization, each C-tree should be configured in such a way that all two distinct children of a node do not occupy the same space as much as possible.

3 Collision Detection Method

Under the assumption that most objects in virtual worlds are stationary or move slowly, the best method for collision detection is to detect static interpenetration between the objects of interest at regular time intervals. Although this collision detection method is not sound, that is, it may fail to detect the collision, it is simpler and faster than the general collision detection, and there are many cases in which this method is suitable under the above assumption. Therefore, it is important to develop the fast interpenetration detection methods.

We use two types of geometrical shapes, spheres and convex polyhedra for the collision detection. Then there are tree types of the interpenetration detection problems: SS-type, PS-type, and PP-type. The *SS-type* problem is to test whether two spheres interpenetrate or not, *PS-type* refers whether a convex polyhedron and a sphere interpenetrate or not, and *PP-type* deals with two convex polyhedra.

The SS-type and PS-type problems are easy to solve. The PP-type problem can be formulated as the linear programming problem [8]. We use the simplex method to solve the PP-type problem. The simplex method is an algorithm which solves the linear programming problems. One of the striking features of the simplex method is that, while its worst case behavior is known to be exponential with respect to the dimension of the solution space, its expected behavior is almost always excellent in practice [7]. Moreover the simplex method can provide additional information such as the collision point and the collision time.

4 Experiments

Our implementational goal is to provide the users with the ability to manipulate directly computer-generated objects in realtime. The graphically modeled 3D hand object is named as *Glove-Cursor*(GC).

Our collision detection scheme is implemented in *C++ language* on *PC 386DX*. We use the basic rendering part of *Rend386* floating-point version for 3D object rendering. The GC is implemented as a hierarchical object with 12 segments: 2 segments for each finger,

1 segment for the palm and back of the hand, and 1 segment for the frame of reference to which the position and orientation of GC are defined. *Nintendo Power-Glove*(PG) is used as the device controlling GC. Our experiment is conducted on the two dimensional computer screen. Our programs, however, can be easily ported to the 3D display such as the head-mounted-display or three dimensional shutter glasses since our rendering package assumes a continuously moving viewpoint.

We test the proposed collision detection scheme with the virtual human figure. The pose and the orientation of the human figure is adjusted by the glove-cursor. In Figures 3 and 4, GC is re-orienting the human's left arm. Figure 5 presents the final configuration where the human is about to throw something out.

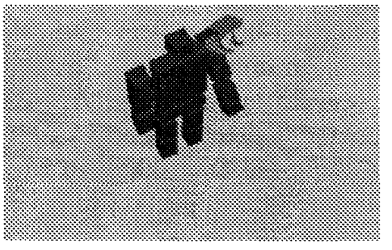

Figure 3:

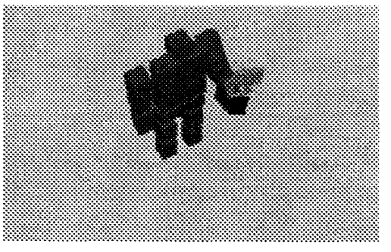

Figure 4:

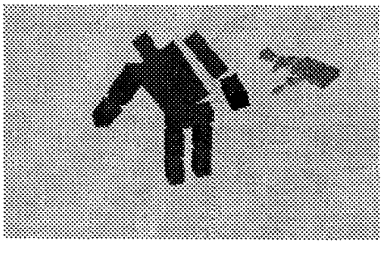

Figure 5:

5 Conclusion

The merit about VR is that a human can undergo computer-generated virtual worlds which are difficult or impossible to experience in the real world. However, unless the virtual world was equipped with collision detection function, all objects could penetrate each other, that is, they would behave like ghosts. Thus, for a true VR, the collision detection is inevitable.

There has been little systematic attempt to localize the collision regions between graphical objects so far. Our C-tree enables us to localize the possible collision regions between two three dimensional hierarchical objects efficiently, which allows our collision detection scheme to be utilized to implement more realistic VR system and to make users directly manipulate the three dimensional hierarchical objects in real time.

References

[1] Stephen Cameron, "Collision Detection by Four-Dimensional Intersection Testing," IEEE Transactions on Robotics and Automation, Vol.6, No.3, 1990.

[2] John Canny, "Collision Detection for Moving Polyhedra," IEEE Transactions on Pattern Analysis and Machine, Vol. Pami-8, No.2, 1986.

[3] R.K. Culley and K.G. Kempf, "A Collision Detection Algorithm based on Velocity and Distance Bounds," Proc. Conf. Robotics. Automat. 1986.

[4] David P. Dobkin and David G. Kirkpatrick, "A Linear Algorithm for Determining the separation of Convex Polyhedra," Journal of Algorithms, 1985.

[5] Vincent Hayward, "Fast Collision Detection Scheme by Recursive Decomposition of A Manipulator Workspace," Proc. Conf. Robotics. Automat. 1986.

[6] Brian Von Herzen, Alan H. Barr, and Harald R. Zatz, "Geometric Collisions for Time-Dependent Parametric Surfaces," Computer Graphics, Vol. 24, No. 4, ACM SIGGRAPH'90 Conference Proceedings.

[7] David G. Luenberger, Linear and Nonlinear Programming. 2nd ed, Addison-Wesley, 1984.

[8] Walter Meyer, "Distances between boxes: applications to collision detection and clipping," Proc. Conf. Robotics. Automat. 1986.

[9] M. Moore and J. Williams, "Collision Detection and Response for Computer Animation," Computer Graphics, Vol. 22, No. 4, ACM, Proceeding of SIGGRAPH'88.